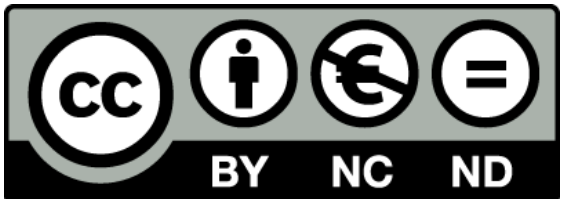
Laboratorio di Programmazione di Sistema

Programma Memorizzato

Luca Forlizzi, Ph.D.

Versione 20.1



Concetto di Programma Memorizzato

- La abstract machine di un *HLL* imperativo è un dispositivo in grado di eseguire programmi su dati
- Nei modelli computazionali degli *HLL*, è ben chiaro che i dati sono memorizzati in "contenitori" accessibili alla abstract machine
- Al contrario, tali modelli non descrivono il modo in cui il dispositivo accede alle istruzioni che formano il programma
- In linea di principio, si può pensare che la abstract machine di un *HLL* imperativo contenga al suo interno il programma da eseguire

Concetto di Programma Memorizzato

- Come sappiamo, in realtà non è così
- Uno dei pregi più importanti dei computer in uso oggi è che sono dispositivi assai flessibili in quanto eseguono programmi che hanno la forma di *software*
- Grazie a ciò, uno stesso computer può eseguire tanti programmi, diversissimi tra loro
- Cambiare il programma eseguito da un computer ha costo pressoché nullo

Concetto di Programma Memorizzato

- In questa presentazione usiamo le seguenti definizioni:
  - M_4: generica abstract machine di livello 4, in grado di eseguire programmi *ASM*
  - M_2: generica abstract machine di livello 2, in grado di eseguire programmi *LM*
- Come è noto, M_2 esegue istruzioni che sono contenute nella memoria principale, che è esterna a M_2
- Questo è il concetto di *programma memorizzato*

Concetto di Programma Memorizzato

- In particolare nel modello di Von Neumann
  - Le istruzioni sono memorizzate nella memoria principale, che è usata anche per memorizzare i dati
  - Le istruzioni sono sequenze di stringhe binarie, così come i dati
- Di norma, la memoria immagazzina bit senza attribuire loro un significato: dunque non distingue tra istruzioni e dati (così come non distingue tra tipi di dato diversi)
- Al contrario, M_2 interpreta i bit letti dalla memoria, in certi casi come istruzioni e in altri come dati
- Il concetto di programma memorizzato è presente in *ASM*, in modo simile a come è presente nel livello 2
- Ciò marca ulteriormente la differenza tra l'*ASM* e i linguaggi *HLL*

Programma Memorizzato in una *ISA*

- Riepiloghiamo come una abstract machine di livello 2 M_2 preleva dalla memoria e ed esegue le istruzioni
- Ogni istruzione del LM è una stringa di valori binari, immagazzinata in un'area di memoria
- Chiamiamo *indirizzo* di un'istruzione, l'indirizzo dell'area di memoria che la contiene
- Chiamiamo *dimensione* di un'istruzione, la dimensione, espressa in byte, dell'area di memoria che contiene l'istruzione

Programma Memorizzato in una *ISA*

- L'esecuzione di un'istruzione da parte di M_2 inizia con la lettura dell'istruzione stessa dalla memoria (fase di *fetch*)
- A tale scopo, M_2 dispone di un registro speciale chiamato *program counter* (PC) che contiene l'indirizzo dell'istruzione
  - Si tratta di una denominazione non particolarmente felice, ma che è ormai di uso comune; alcuni autori propongono come denominazione più appropriata *Instruction Address Register*
- Dopo aver letto i bit che formano l'istruzione, M_2 ne interpreta il significato (fase di *decode*) e mette in atto le azioni necessarie per effettuare le operazioni previste dall'istruzione (fase di *execute*)

Programma Memorizzato in una *ISA*

- Durante l'esecuzione dell'istruzione, il PC viene modificato in modo da contenere l'indirizzo da cui leggere l'istruzione successiva in ordine di esecuzione
- Le istruzioni, con l'esclusione di quelle di salto, modificano il PC aggiungendo ad esso la dimensione dell'istruzione corrente
- In questo modo la prossima istruzione eseguita avrà indirizzo pari al valore dell'indirizzo dell'istruzione corrente sommato con la dimensione dell'istruzione corrente

Programma Memorizzato in una *ISA*

- In altre parole, la prossima istruzione eseguita avrà come indirizzo quello successivo all'indirizzo massimo dell'area di memoria che contiene l'istruzione corrente
- Quindi non ci sono "buchi" tra l'area in cui è memorizzata l'istruzione corrente e quella in cui è memorizzata l'istruzione successiva
- Le istruzioni di salto, invece, possono modificare in modo diverso il PC, in base alla propria semantica

Programma Memorizzato in un *ASM-PM*

- Il concetto di programma memorizzato emerge al livello 4 per via della corrispondenza tra le istruzioni che formano un programma *ASM* e le istruzioni che formano la traduzione in *LM* di tale programma
- Ogni istruzione *ASM* ha una traduzione costituita da una o più istruzioni *LM*, e quindi ha:
  - Un *indirizzo*, pari all'indirizzo dell'area di memoria che contiene la sua traduzione *LM*
  - Una *dimensione*, pari alla dimensione dell'area di memoria che contiene la sua traduzione *LM*
- Il registro PC è accessibile anche da una abstract machine di livello 4 M_4

Programma Memorizzato in un *ASM-PM*

- Gli indirizzi delle istruzioni *ASM* vengono stabiliti dall'assembler, quando esso opera in modalità *Assemble-go*, oppure dal loader, al momento del caricamento, sulla base delle informazioni predisposte dall'assembler
- L'assegnazione degli indirizzi alle istruzioni viene fatta in modo che le loro traduzioni *LM* vengano eseguite da M_2 nell'ordine previsto nel programma *ASM*
- Come sappiamo, le istruzioni *ASM*, con l'esclusione di quelle di salto, vengono eseguite da M_4 nell'ordine in cui compaiono nel codice sorgente (ordine testuale)
- Come detto in precedenza, le istruzioni *LM*, con l'esclusione di quelle di salto, vengono eseguite da M_2 in ordine di indirizzi crescenti

Programma Memorizzato in un *ASM-PM*

- Tenendo presente il modo in cui M_2 modifica il PC durante l'esecuzione delle istruzioni, per far corrispondere l'ordine testuale del codice sorgente *ASM* con l'ordine di esecuzione della traduzione *LM* di tale codice, si deve usare la regola seguente:
- Se I_1, I_2 sono due istruzioni *ASM* e I_2 è il successore di I_1 in ordine testuale, l'indirizzo di I_2 deve essere pari a l'indirizzo di I_1 aumentato della dimensione di I_1

Indirizzamento delle Istruzioni

- In un programma *ASM*, si può indicare l'indirizzo di un'istruzione tramite una label, allo stesso modo in cui si usano le label per indicare gli indirizzi delle parole di memoria che contengono dati
- Una label seguita immediatamente da un'istruzione, ha il valore dell'indirizzo di tale istruzione
- Inoltre, conoscendo quali sono le dimensioni delle istruzioni, l'indirizzo di una istruzione può essere calcolato a partire dall'indirizzo di altre istruzioni

Indirizzamento delle Istruzioni

- Facciamo alcuni esempi, considerando una sequenza di 3 istruzioni i_1, i_2, i_3
  - Se conosciamo l'indirizzo di i_1 e le dimensioni di i_1 e i_2, possiamo calcolare, sommando tali dati, l'indirizzo di i_3
  - Se invece conosciamo l'indirizzo di i_3, possiamo calcolare l'indirizzo di i_1, sottraendo dall'indirizzo di i_3, le dimensioni di l_1 ed l_2
  - Se conosciamo l'indirizzo di i_2, possiamo usarlo per calcolare quello di i_1 (sottraendo la dimensione di i_1 dall'indirizzo di i_2) e quello di i_3 (aggiungendo la dimensione di i_2 all'indirizzo di i_2)

Indirizzamento delle Istruzioni

- Ogni istruzione di salto individua un'istruzione di destinazione del salto
  - Ciò viene fatto calcolando l'indirizzo dell'istruzione destinazione del salto
  - Tale calcolo è di solito statico per le istruzioni di salto con destinazione statica, mentre è necessariamente dinamico per le istruzioni di salto con destinazione dinamica

Indirizzamento delle Istruzioni

- Ogni istruzione di salto (dopo aver stabilito se il salto deve avvenire o meno, nel caso di istruzioni condizionate), *indirizza* la prossima istruzione da eseguire, ovvero
  - calcola l'indirizzo dell'istruzione di destinazione del salto, nel caso di destinazione dinamica
  - copia l'indirizzo dell'istruzione di destinazione del salto nel PC
- L'indirizzamento della prossima istruzione da eseguire, viene fatto in base ad un modo di indirizzamento, in maniera analoga a quanto avviene per i dati

Indirizzamento delle Istruzioni

- La traduzione *LM* di un'istruzione *ASM* è il contenuto di un'area di memoria, corrispondente a una o più parole di memoria
- Pertanto tutti i modi di indirizzamento adatti a dati in memoria, introdotti in precedenti presentazioni, possono essere usati come modi di indirizzamento per istruzioni
- Nel caso di istruzioni di salto con destinazione dinamica, è necessario utilizzare un modo di indirizzamento che permetta di calcolare e modificare dinamicamente l'indirizzo dell'operando, ad esempio un modo di indirizzamento indiretto-memoria
- Un ulteriore tipologia di modi di indirizzamento, molto comune in relazione alle istruzioni, è quella dei modi di indirizzamento *PC-indicizzato*

Indirizzamento PC-indicizzato

- L'indirizzamento *PC-indicizzato* funziona in modo simile all'indirizzamento indicizzato, con la differenza che il registro base è il PC invece che un general purpose register
- In un modo di indirizzamento *PC-indicizzato*, l'operando è una parola di memoria il cui indirizzo è la somma del contenuto del PC e di una costante intera, detta *offset*
- Nell'indirizzamento di istruzioni, l'offset indica, in pratica, di quanto aumentare o diminuire il PC per indirizzare la prossima istruzione da eseguire

Indirizzamento PC-indicizzato

- L'indirizzamento avviene al momento dell'esecuzione dell'istruzione *I* che usa il modo di indirizzamento PC-indicizzato: pertanto il contenuto del PC è pari all'indirizzo di *I*, aumentato di un determinato valore costante che dipende dallo specifico *ASM-PM*, per motivi legati al funzionamento interno delle implementazioni della abstract machine
- Poiché, di norma un'istruzione ha sempre lo stesso indirizzo, l'indirizzamento PC-indicizzato calcola sempre lo stesso indirizzo per l'operando, e quindi può essere usato solo per istruzioni di salto con destinazione statica
- L'indirizzamento PC-indicizzato è vantaggioso rispetto ad altri, nel caso in cui gli indirizzi delle istruzioni che lo usano vengano modificati, come accade spesso nei computer che usano un sistema operativo

Indirizzamento PC-indicizzato

- In alcuni *ASM-PM*, l'indirizzamento PC-indicizzato può essere usato anche per accedere a dati
- Ad esempio in MC68000 il modo di indirizzamento PC-indicizzato può essere usato da molte istruzioni di trasferimento dati e aritmetico-logiche (ad esempio da `move` o da `add`), ma solo per gli operandi che non vengono modificati dall'istruzione

Indirizzamento delle Istruzioni in MIPS32-MARS

- L'istruzione `b` e tutte le istruzioni di salto condizionato usano il modo di indirizzamento PC-indicizzato
- L'indirizzo dell'istruzione di destinazione del salto (statico) deve essere obbligatoriamente espresso da una label
- MARS calcola automaticamente, a partire dalla label, la costante numerica che viene sommata al valore che sarà contenuto in PC al momento dell'esecuzione dell'istruzione

Indirizzamento delle Istruzioni in MIPS32-MARS

- Sia *I* l'istruzione `b` o un'istruzione di salto condizionato, e sia *L* la label che esprime l'indirizzo di destinazione di *I*
- Per motivi legati alla sintassi delle traduzioni in *LM* di `b` e delle istruzioni di salto condizionato, il valore assoluto della differenza tra l'indirizzo rappresentato da *L* e l'indirizzo di *I* deve essere minore o uguale di $2^{15} - 1$
- In altre parole, `b` e le istruzioni di salto condizionato possono saltare solo a istruzioni relativamente "vicine" in memoria

Indirizzamento delle Istruzioni in MIPS32-MARS

- L'istruzione `j` utilizza l'indirizzamento diretto-memoria
- L'indirizzo dell'istruzione di destinazione del salto (statico) deve essere obbligatoriamente espresso da una label
- Il modo di indirizzamento diretto-memoria usato da `j` è però diverso da quello usato dalle istruzioni di trasferimento dati di MIPS32
- In particolare, il modo di indirizzamento diretto-memoria usato da `j` specifica solo una parte dell'indirizzo di destinazione del salto: infatti la label deve obbligatoriamente indicare un'indirizzo che ha i 4 bit più significativi uguali al contenuto del PC al momento dell'esecuzione di `j`

Indirizzamento delle Istruzioni in MIPS32-MARS

- MIPS32 ha anche un'istruzione di salto incondizionato con destinazione *dinamica*
- Si tratta dell'istruzione `jr`, che ha come unico operando uno dei GPR
- Tale istruzione usa il modo di indirizzamento indiretto-registro: l'indirizzo di destinazione del salto è il contenuto dell'operando registro

Indirizzamento delle Istruzioni in MC68000-ASM1

- In MC68000, quando viene eseguita un'istruzione *I* che usa il modo di indirizzamento PC-indicizzato, PC contiene l'indirizzo di *I* aumentato di 2
- L'istruzione `bra` e le istruzioni di salto condizionato usano il modo di indirizzamento PC-indicizzato
- L'operando di tali istruzioni può essere una costante numerica oppure una label

Indirizzamento delle Istruzioni in MC68000-ASM1

- Sia *I* un'istruzione `bra` o un'istruzione di salto condizionato
  - Se l'operando di *I* è una costante numerica *k*, allora *k* è l'offset che viene sommato al contenuto di PC per indirizzare l'istruzione di destinazione del salto
  - Se l'operando di *I* è una label *L*, l'offset viene calcolato da ASM-One, a partire dall'indirizzo rappresentato da *L* e dall'indirizzo di *I*
- Per motivi legati alla sintassi delle traduzioni in *LM* di `bra` e delle istruzioni condizionato, il valore assoluto dell'offset deve essere minore o uguale di $2^{15} - 1$
- Dunque, `bra` e le istruzioni di salto condizionato possono saltare solo a istruzioni relativamente "vicine" in memoria

Indirizzamento delle Istruzioni in MC68000-ASM1

- L'istruzione `jmp` può usare diversi modi di indirizzamento per dati in memoria
- In LPS ci limitiamo a considerare i più usati, ovvero diretto-memoria e indiretto-registro
- Tramite l'indirizzamento diretto-memoria, si può specificare, staticamente, l'intero indirizzo di destinazione del salto
- Quindi `jmp` può saltare, incondizionatamente, ad un'istruzione che può trovarsi ad un qualunque indirizzo, anche lontano
- Usando l'indirizzamento indiretto-registro, si specifica che l'indirizzo di destinazione del salto è il contenuto di un registro indirizzi
- Quindi `jmp`, usata con tale modo di indirizzamento, effettua un salto incondizionato con destinazione dinamica ad un qualunque indirizzo, anche lontano

Istruzioni come Dati

- Il fatto che le istruzioni siano memorizzate e abbiano indirizzi, rende possibile scrivere in *ASM* programmi che utilizzano come dati gli indirizzi delle istruzioni o addirittura le stesse traduzioni *LM* delle istruzioni del programma
- Ciò permette di realizzare
  - salti con destinazione dinamica
  - salti ad istruzioni i cui indirizzi sono calcolati dinamicamente con operazioni aritmetiche
  - programmi che copiano e modificano altri programmi
- L'esempio *Switch* mostra come, mediante salti con destinazione dinamica, sia possibile effettuare in modo efficiente una selezione con molte alternative, come nell'istruzione `switch` di C Standard